Hochschule Harz	FB Automatisierung und Informatik
1. Labor: Grafikdarstellung	Grafische Nutzerschnittstellen mit Java Thema:Anzeigen einer Grafikdatei

Versuchsziele

Vertiefung im Verständnis der Dialog-Programmierung, Benutzung von Klassen für die Grafikanzeige.

Aufgabenstellung:

Als Grundlage gibt es ein fertiges „Register-Projekt“. Erweitern Sie dieses mit den Grafikfunktionen. Ein Register soll die ausgewählte Datei über eine Klasse Middleware einlesen und über die Klasse „Canvas“ darstellen.

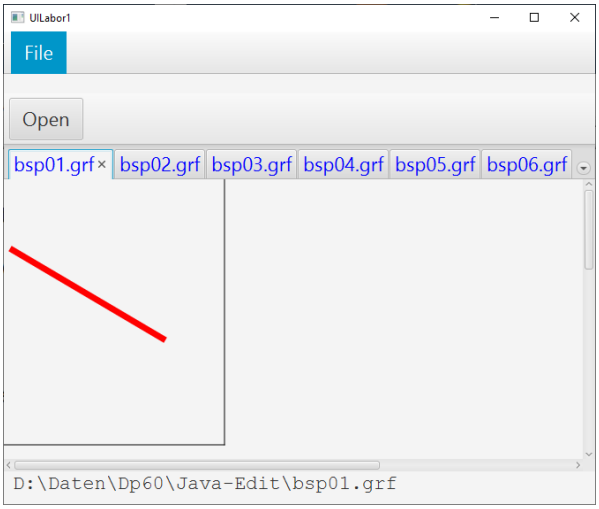


Abbildung 1 Musterlösung

Wichtige Hinweise:

- Jedes Register zeigt eine Datei grafisch dar.
- Es werden nur vorhandene Dateien benutzt.
- **Mit dem „Open-File-Event“ wird eine Datei oder mehrere Dateien ausgewählt und in jeweils ein neues Register eingetragen.**
- Gesamte Musterlösung (inkl. Vorlabor): 750 Zeilen
- Einige vorhandene Quellcodes müssen geändert bzw. gelöscht werden

Aufgabenstellung:

Entwickeln Sie ein Programm, welches Grafikdateien (*.grf) nach der unten aufgelisteten Spezifikation einliest und die darin gespeicherten Objekte auf einem Canvas eines Registers grafisch darstellt.

Die Datei gliedert sich in zwei Teile. Der erste Teil besteht aus einem Kopfteil (Header), der zweite besteht aus den grafischen Elementen. In der Einleseroutine wird erst der Header gelesen und dann in einer Schleife die verschiedenen Objekte. Für jeden Datensatz muss ein passendes Objekt erzeugt werden und in eine interne ArrayList der Klasse „Middleware“ eingetragen werden (Ein Beispiel ist vorgegeben). In der „Paint-Methode“ der Klasse „Canvas“ werden die Objekte gezeichnet. **Die Klasse „Canvas“ speichert die Middleware als Referenz.**

Für jedes grafisches Objekt ist eine eigene Klasse zu entwerfen (z. B. MyLine, MyCircle, MyTriangle, MyPolyline). Dabei sind sämtliche Attribute als „privat“ zu deklarieren.

Für die Aufgabe werden die benötigten Klassen in einer Rohform zur Verfügung gestellt. Weitere Information sind den Skripten bzw. den Beispieldateien zu entnehmen.

Kommentierung Sie den Quelltext. Switch-Anweisungen müssen den default-Fall abprüfen und mit einer Fehlermeldung abschließen (passive Programmierung).

Wichtig:
Fehlerhafte Dateien dürfen das Programm nicht zum Absturz bringen. Das Programm soll dann eine sinnvolle Fehlermeldung ausgeben. Dazu soll in der Klasse „Middleware“ eine private Variable mit get-Methode implementiert werden. Diese zeigt an, ob das Einlesen vollständig war. Sinnvoll wäre zum Beispiel ein String, der mit einem Fehlertext versehen werden kann. Diese Text könnte dann im Canvas angezeigt werden.

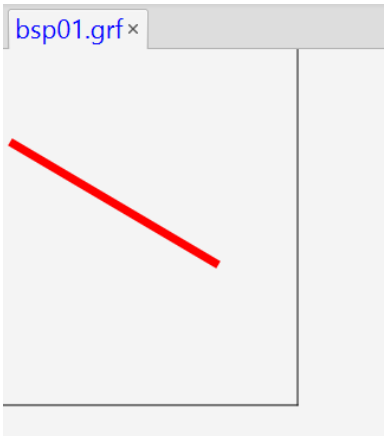
Hinweis:
Es gibt fehlerhafte Dateien, die **auch** abgefangen werden müssen!

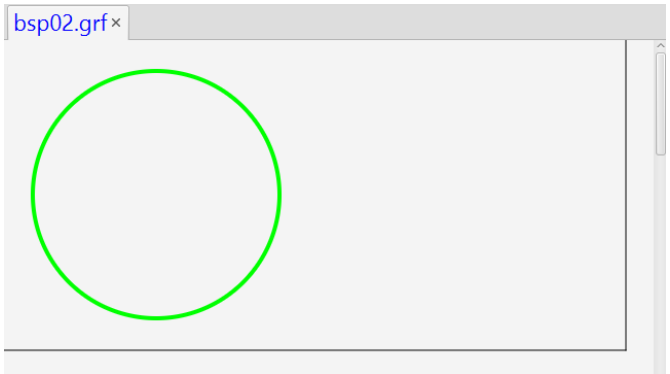
Inhalt der Beispieldateien:

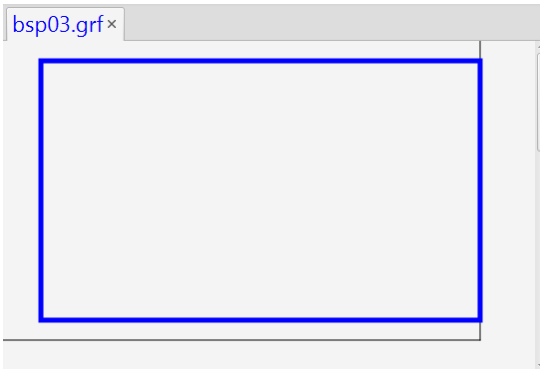
bsp01.grf	Linie
bsp02.grf	Kreis
bsp03.grf	Rechteck
bsp04.grf	Kreis, Quadrat
bsp05.grf	Zwei Polylinien
bsp06.grf	zwei Dreiecke
bsp07.grf	Würfel bestehend aus 1 Rechteck, 2 Linien, 1 Polyline
bsp08.grf	zwei Rechtecke mit zwei Inkreisen
bsp09.grf	zwei Rechtecke mit Kreisen
bsp10.grf	1000 Linien, jede einzeln eingebaut ;-)
bsp11.grf	Surprise ;-)

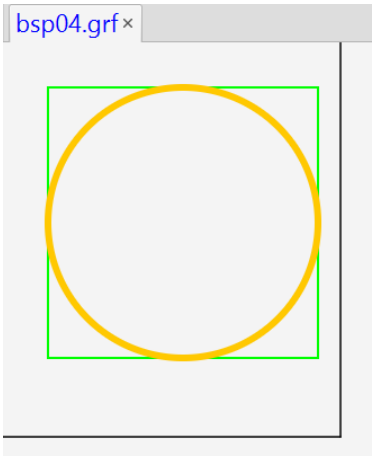
bspError1.grf	Fehlerhafte Datei
bspError2.grf	Fehlerhafte Datei
bspError3.grf	Fehlerhafte Datei
bspError4.grf	Fehlerhafte Datei
bspError5.grf	Fehlerhafte Datei

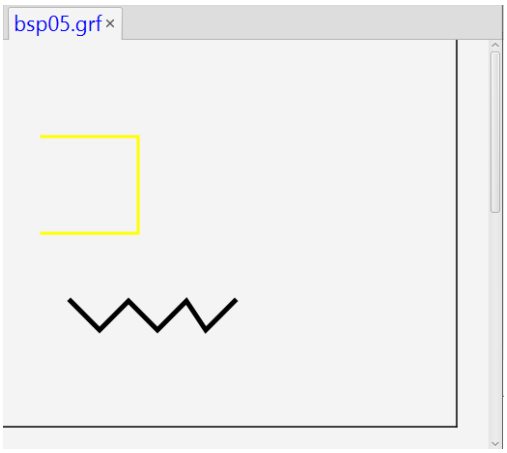
Lösungen der mitgelieferten Dateien mit den Originalen Fenstergrößen:

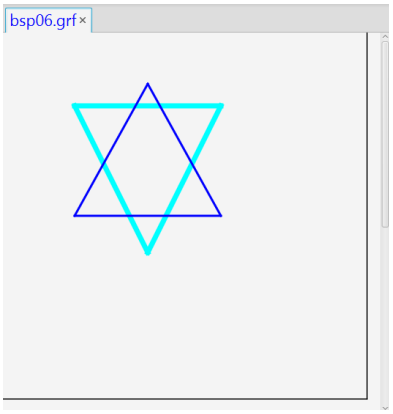


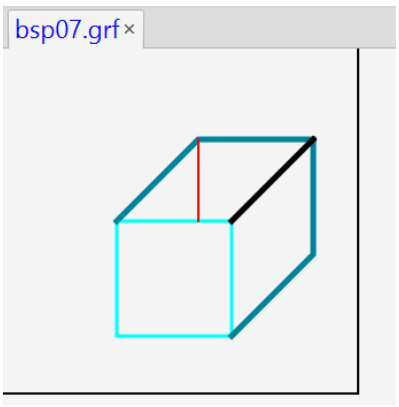


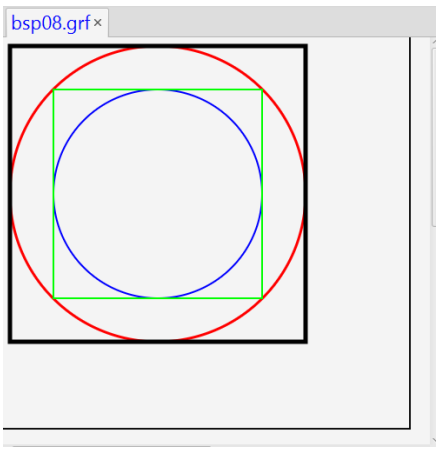

bsp03.grf

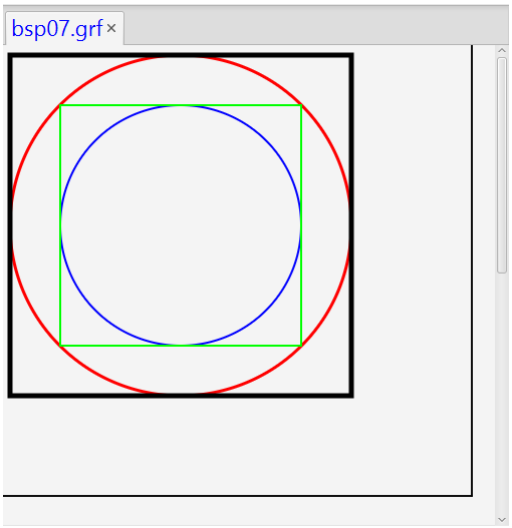

bsp04.grf

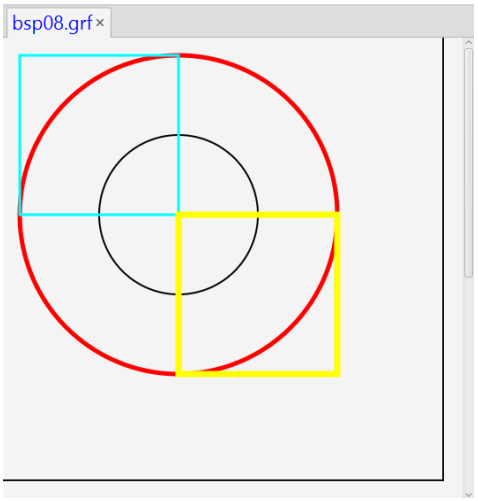

bsp05.grf

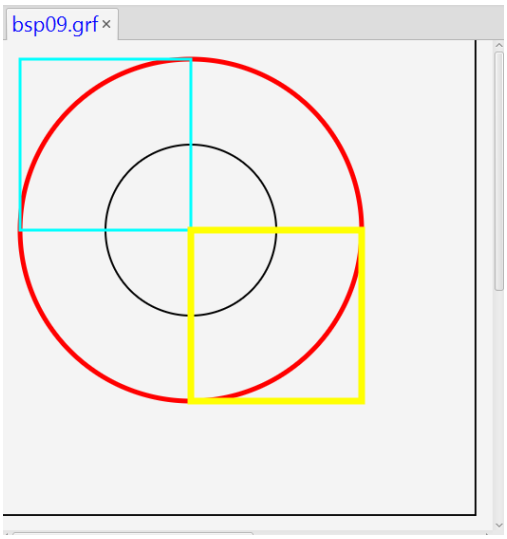

bsp06.grf

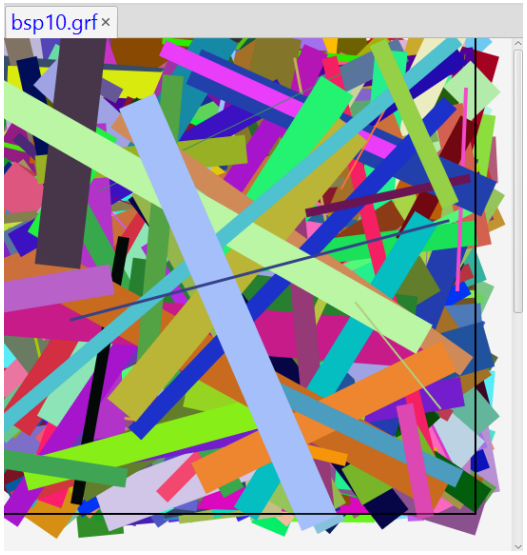

bsp07.grf


bsp08.grf









?

Ausgabe Datei “bsp11.grf”

Typen und Koordinaten der Beispiel-Dateien

BSP01.grf

Fenster: 300 x 250 Pixel (w/h)
Linie: 10/80 bis 180/180
Farbe: Rot
Linienstärke: 2

BSP02.grf

Fenster: 607 x 331 Pixel (w/h)
Kreis: Mittelpunkt: 150/150
Radius: 120
Farbe: Grün
Linienstärke: 4

BSP03.grf

Fenster: 480 x 300 Pixel (w/h)
Rechteck: 40/20 bis 480/280
Farbe: Blau
Linienstärke: 5

BSP04.grf

Fenster: 300 x 350 Pixel (w/h)
Rechteck: 40/40 bis 280/280
Farbe: Grün
Linienstärke: 2
Kreis: Mittelpunkt: 160/160
Radius: 120
Farbe: Rot
Linienstärke: 6

BSP05.grf

Fenster: 470 x 400 Pixel (w/h)
Polylinie: 40,100
140,100
140,200
40,200
Farbe: Gelb
Linienstärke: 3

Polylinie: 70,270
100,300
130,270
160,300
190,270
210,300
240,270
Farbe: Schwarz
Linienstärke: 5

BSP06.grf

Fenster: 500 x 500 Pixel (w/h)
1. Dreieck: 100,100
300, 100
200,300
Farbe: Cyan
Linienstärke: 7
2. Dreieck: 100, 250
300, 250
200,70
Farbe: Blau
Linienstärke: 3

BSP07.grf (Würfel)

Fenster: 310 x 300 Pixel (w/h)
Rechteck: 100/150 bis 200/250
Farbe: Teal
Linienstärke: 3
1. Linie: 100/150 bis 171/79
Farbe: Rot
Linienstärke: 3
Polyline: 100/150
171/79
271/79
271/179
200/250
Farbe: 33435
Linienstärke: 5
2. Linie: 171/79 bis 171/150
Farbe: Rot
Linienstärke: 2
3. Linie: 271/79 bis 200/150
Farbe: Schwarz
Linienstärke: 5

BSP08.grf

Fenster: 450 x 470 Pixel (w/h)
Circle: 180/180, Radius 170
Farbe: Rot
Linienstärke: 3
Rectangle: 10/10 bis 350/350
Farbe: Schwarz
Linienstärke: 5

Circle: 180/180, Radius 120
Farbe: Blau
Linienstärke: 2
Rectangle: 60/60 bis 180/180
Farbe: Grün
Linienstärke: 2

BSP09.grf

Fenster: 500 x 500 Pixel (w/h)
Circle: 200/200, Radius 180
Farbe: Rot
Linienstärke: 5
Circle: 200/200, Radius 90
Farbe:Schwarz
Linienstärke: 2

Rectangle: 20/20 bis 200/200
Farbe: Cyan
Linienstärke: 3
Rectangle: 200/200 bis 380/380
Farbe: Gelb
Linienstärke: 7

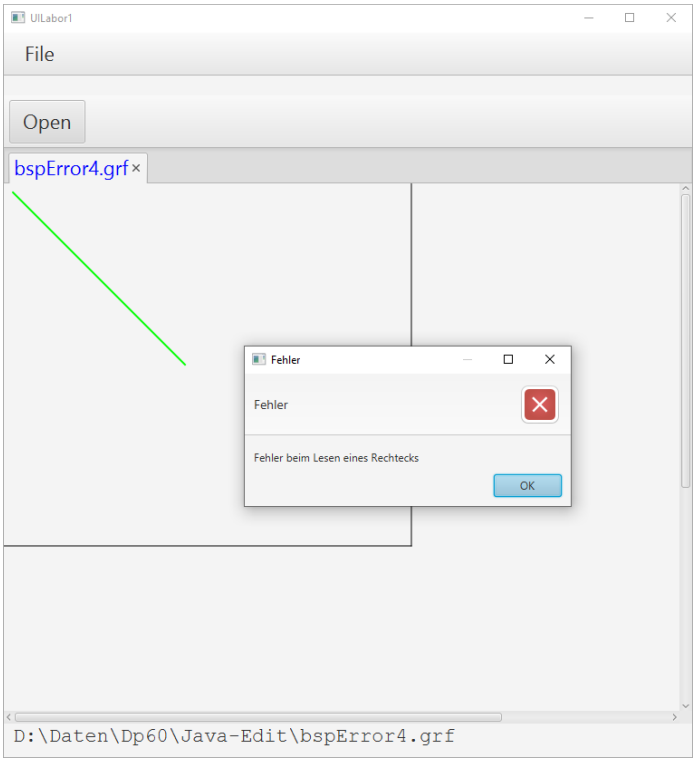
BSP10.grf

Fenster: 500 x 500 Pixel (w/h)
1000 Linien mit Zufalls-Koordinaten und Zufalls-Farben

BSP11.grf: Surprise

Fenster: 500 x 500 Pixel (w/h)
Circle: 250/250, Radius 200
Farbe: Blau
Linienstärke: 2
Rectangle: 120/150 bis 220/175
Farbe: Grün
Linienstärke: 2
Rectangle: 190/150 bis 390/175
Farbe: Grün
Linienstärke: 2
etc.

BSPError1.grf Fehlerhafter Dateikennung ***BSPError2.grf*** Fehlerhafter Header, Version ist 2 ***BSPError3.grf*** Fehlerhafter Header, negative Abmessungen	***BSPError4.grf*** Negative Koordinaten ***BSPError5.grf*** Falsche Kennung



Aufbau der Datei

Die Java-Dateien haben folgenden Aufbau:

Open-File-Event:
Create Register (TabView)
Create Middleware, jede grf-Datei hat ihre eigene Middleware
readFile
Read Header (loadFromData)
Read Objekte (loadFromData)
Show Windows

readFile in der Klasse Middleware:
Read Header
Read Objekt-Kennung, korrekter Typ?
While(...) {
Abfrage bezüglich der Objekt-Kennung
Create Objekt
Read Objekt
Insert Objekt
Read Objekt-Kennung, korrekter Typ?
}

Paint Routine in MyCanvas
For all Objekte in liste {
Referenziere Objekt
gc.paint (…) Polymorphismus
}

Klasse Header
Private Attribute
Set / get -Methoden
loadFromData(...)

Klasse MyLine
Private Attribute
loadFromData(...)
paint(…)

KlasseMyCircle
Private Attribute
loadFromData(...)
paint(…)

etc.

Versuchsdurchführung

1. Projekt

- Erstellen eines neuen Projektes namens „Labor1“
- Entzippen der Datei „Labor1.zip“ (Homepage)
- Erstellen der Klassen und einfügen aus der Vorlage
    - Labor1.txt
    - Basis.txt
    - Konstanten.txt
    - Middleware.txt
    - MyCanvas.txt
    - MyCircle.txt
    - MyGrfObject.txt
    - MyHeader.txt
    - MyLine.txt
    - MyPolyline.txt
    - MyRect.txt
    - MyTriangle.txt

2. Klasse MyCanvas

- Deklaration der Variablen
    - private Middleware middleware;
    - private MyCanvas canvas;
- Konstruktor
    - Initialisieren der Variablen
    - Erstellen einer Instanz der Klasse „Middleware“
        - Eine Instanz der Klasse Middleware enthält den Dateinamen.
- Die Klasse Canvas erhält einen zusätzlichen Parameter (Middleware middleware)

3. Klasse MyHeader

- Erstellen der Klasse MyHeader. Ein Rahmen ist als Txt-Datei vorhanden. Bitte benutzen.
- Löschen des Default-Konstruktors
- Implementieren der Methode loadFromData
    - Lesen der Magic-Number
    - Lesen und speichern der Versionsnummer
    - Lesen und speichern der Breite
    - Lesen und speichern der Höhe

Ablauf:

- Einbau der Attribute.
- Einbau der set- und get-Methoden.
- Implementierung der Methode „loadFromData“
- Eingelesen werden dürfen die Daten des Headers nur in der Methode „loadFromData(…)“.
- Im Konstruktor darf das Einlesen **nicht** stattfinden.
    - Lesen der Magic-Number
    - Lesen der Version
    - Lesen der Breite und Höhe
    - Test der Attribut auf Korrektheit

4. Konstanten

- Bitte in der Methode „readFile“ benutzen.

5. Klasse Middleware

- Deklaration der Variablen
  - private ArrayList grafikliste Typisiert wäre besser ;-)
  - private String filename;
  - private int height=0;
  - private int width=0;
- Konstruktor
  - Initialisieren der Variablen
- **private** readFile(String filename)
  - Öffnen der Datei
  - Lesen des Headers
  - **Bis hier erstmal testen**

6. Klasse MyGrfObject

- Hat folgende Methoden:
  - void paint(GraphicsContext gc);
  - boolean loadFromData(DataInputStream DataIn);
- Außerdem müssen noch einige Wörter in die Klasse eingefügt werden
- Variablen (interne?):
  - **private** Color c;
  - **private** int linewidth;

7. Klasse MyLine

- Erstellen der privaten Attribute
- Implementieren der abstrakten Methoden
  - loadFromData
    - Lesen von x1, y1, x2, y2
    - Lesen von color (int)
    - Lesen der Linienstärke (linewidth)
  - Implementierung der Methode „paint“
    - setzen der Farbe
    - setzen der Linienstärke
    - zeichnen der Linie

8. Klasse Middleware

- **private readFile(String filename)**
  - **2. Teil:**
  -
  - Speichern der Abmessungen aus dem MyHeader
  - Irgendetwas mit der Grafikliste machen
  - Lesen der Grafikdaten bis zum letzten Datensatz (Typ=0)
    - Abfragen des Typs
    - Erzeugen des Grafiktyps
    - Lesen des Grafiktyps, aber nicht hier
    - Einfügen
  - Schließen des Dateikanals (fin oder din)

9. Klasse MyCanvas

- Erstellen eines Konstruktors mit einer Variablen aus der Klasse Middleware oder die Instanz der Klasse Middleware.
    - Aus dem Parameter wird eine „Liste“ geholt und global gespeichert.
- Überschreiben der Methode „void paint (GraphicsContext gc)“
    - Einen weißen Hintergrund erstellen.
    - Aus der Liste die grafischen Elemente holen.
    - Zeichnen des Rahmens (Breite und Höhe)

Canvas-Methoden (siehe auch Seite 15):

- setFill(Color)
- **fillRect**(double x, double y, double w, double h)
- setStroke(Color) // Linienfarbe
- **strokeLine**(double x1, double y1, double x2, double y2)

9. Testen

- Testen, ob die Linie der ersten Datei angezeigt wird.

Test mit der ersten Datei **bsp01.grf**

- Fenster: 300 x 250 Pixel (w/h)
- Linie: 10/80 bis 180/180
- Farbe: Rot
- Linienstärke: 2

10. Klasse MyCircle

- Erstellen der privaten Attribute
- Implementieren der abstrakten Methoden
    - loadFromData
        - Lesen von x, y, radius
        - Lesen von color (int)
        - Lesen der Linienstärke (linewidth)
    - paint
        - setzen der Farbe
        - setzen der Linienstärke
        - zeichnen des Kreises

11. Klasse MyRect

- Erstellen der privaten Attribute
    - Width und Height dürfen nicht als Attribute verwendet werden.
- Implementieren der abstrakten Methoden
    - loadFromData
        - Lesen von x1, y1, x2, y2
            - Width und Height dürfen nicht als Attribute verwendet werden.
        - Lesen von color (int)
        - Lesen der Linienstärke (linewidth)
    - paint
        - setzen der Farbe
        - setzen der Linienstärke
        - zeichnen des Rechteckes

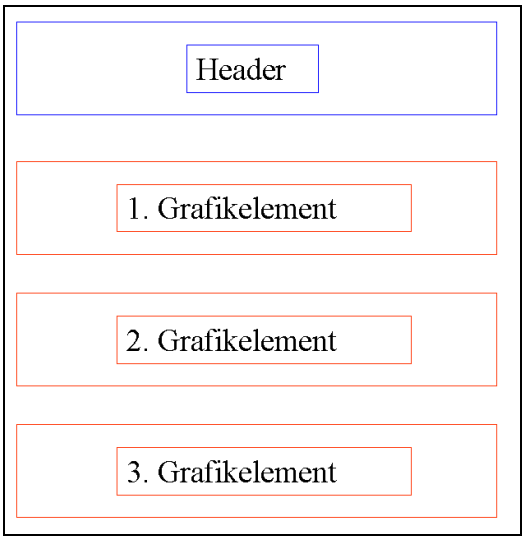
13. Klasse MyTriangle

- Erstellen der privaten Attribute
- Implementieren der abstrakten Methoden
    - loadFromData
        - Lesen von x1, y1, x2, y2, x3, y3
        - Lesen von color (int)
        - Lesen der Linienstärke (linewidth)
    - paint
        - setzen der Farbe
        - setzen der Linienstärke
        - setzen der drei einzelnen Linien

13. Klasse MyPolyline

- Erstellen der privaten Attribute
    - Farbe
    - Array von x
    - Array von y
- Implementieren der abstrakten Methoden
    - loadFromData
        - Lesen der Anzahl der Koordinaten
        - Irgendetwas erzeugen
        - Lesen der Koordinaten
        - Lesen der Farbe
        - Lesen der Linienstärke (linewidth)
    - paint
        - setzen der Farbe
        - setzen der Linienstärke
        - zeichnen der Polyline

Aufbau der Dateien:



Aufbau des Headers:

Byte	Bedeutung	Erläuterung
0-1	Dateikennung	KI (KommunkationsInformatik) K=75 I=73
2-3	Version	1
4-7	Breite der Grafik	Bedingung: Breite > 0
8-11	Höhe der Grafik	Bedingung: Höhe> 0

Definition der Konstanten:

1:	Linie	x1, y1, x2, y2, Farbe, Linienstärke
2:	Rechteck	x1, y1, x2, y2, Farbe
3:	Kreis	x1, y1, Radius, Farbe
4:	Dreieck	x1, y1, x2 y2, x3, y3, Farbe
5:	Polylinie	x1, y1, …, xn, yn, Farbe

Schema der Speicherung der Objekte:

Header

Kennung des 1. Objektes (2 Byte)
Datensatz des 1. Objektes

Kennung des 2. Objektes (2 Byte)
Datensatz des 2. Objektes

Kennung des 3. Objektes (2 Byte)
Datensatz des 3. Objektes

Kennung des n. Objektes (2 Byte)
Datensatz des n. Objektes

Kennung 0 ⇒ **DATENSATZENDE**

Definition der Linie (MyLine):

Bytes	Type
0-1	Kennung (1)
2-5	x1
6-9	y1
10-13	x2
14-17	y2
18-21	Farbe
22-25	Strichstärke

Definition des Rechtecks (MyRect):

Bytes	Type
0-1	Kennung (2)
2-5	x1
6-9	y1
10-13	x2
14-17	y2
18-21	Farbe

Definition des Kreises (MyCircle):

Bytes	Type
0-1	Kennung (3)
2-5	x
6-9	y
10-13	Radius
14-17	Farbe

Definition des Dreiecks (MyTriangle):

Bytes	Type
0-1	Kennung (4)
2-5	x1
6-9	y1
10-13	x2
14-17	y2
18-21	x3
22-26	y3
27-30	Farbe

Definition der Polylinie (MyPolylinie):

Bytes	Type
0-1	Kennung (5)
2-5	Anzahl der Koordinaten (0 bis beliebig)
6-9	x1
10-13	y1
14-17	x2
18-21	y2
22-25	x3
26-29	y3
30-33	x3
34-37	y3
etc.	
38-41	Farbe

Grafikfunktionen in Java:

Methode	Beschreibung:
strokeLine()	zeichnet eine Gerade *void strokeLine(int x1, int y1, int x2, int y2)*
strokeOval()	zeichnet eine Ellipse *void strokeOval(int x, int y, int width, int height)*
strokePolyline	zeichnet ein Linie mit mehreren Punkten (Array) *void* stroke*Polyline(int [] xPoints, int [] yPoints, int anz)*
strokeRect()	zeichnet ein Rechteck *void strokeRect(int x, int y, int width, int height)*
setStroke()	legt die Linien**farbe** fest gc.setStroke(Color.RED); *int color = din.readInt(); // rgb* *int r=(color ...)* *int g=(color ...)* *intbr=(color ...)* c = Color.rgb(r,g,b); // Color.web(color.toHexString(color))
fillRect	zeichnet ein Rechteck mit der aktuellen Füllfarbe *fillRect(double x, double y, double w, double h)*
setFill()	setzt die Füllfarbe *setFill(Color)*
setLineWidth	Strichdicke setzen *setLineWidth(linewidth);*

Hinweise:

Farben

Die gespeicherten Farben entsprechen einem RGB-Wert (32-Bits).
Das Objekt „Color“ wird durch folgenden Konstruktor erzeugt:

- Color c = Color.GREEN;
- Color c = Color.rgb(0, 255, 0) ; // (Color.green);

Beispielcode:

Color c;

int rwert = 0;
intgrwert = 255;
int bwert = 0;
c = Color.rgb(rwert, gwert, bwert) ;

Das Abspalten der einzelnen Farbkomponenten aus einem int-Wert geschieht über logische und Shift-Funktionen.

- `int r=(color&0xFF0000)>>??;  // jeweils an den rechten Anfang`
- `int g=(color&0xFF00)>>??;  // jeweils an den rechten Anfang`
- `int b=(color&0xFF);`

Change-Listener eines TabViews

```
tabpane.getSelectionModel().selectedItemProperty().addListener(
  new ChangeListener<Tab>() {
    @Override
    public void changed(ObservableValue<? extends Tab> ov,Tab told, Tab tnew) {
       if (tnew!=null) {
          // Aktion
       } // if
    }
  }
);
```

Einfügen eines Tab in einem TabViews

```
private void  insertTab(String filename) {
    MyTextArea editor = new MyTextArea(filename);
    String caption="noname";
    if (filename.length()>0) caption=getFileNameExt(filename);
    Tab tab = new Tab(caption);
    tab.setTooltip(new Tooltip(filename));    // javafx.scene.control.Tooltip
    tab.setContent(new ScrollPane(canvas));
    tabpane.getTabs().add(tab);
     // eventuell hier zeichnen
  }
```

String getFileNameExt(String filename)

```
public String getFileNameExt(String filename) {
  int k, n;
  n = filename.length();
  k = filename.lastIndexOf("\\");
  if (k == -1) {
    // test mit /
    k = filename.lastIndexOf("/");
    if (k == -1)
      return filename;
    else {
       // beginIndex   endIndex
      return filename.substring(k+1, n);
    }
  }
  else {
    // beginIndex   endIndex
    return filename.substring(k+1, n); // ??????  n-1 ist zu klein
  }
}  // getFileNameExt
```

Beispiele:

Einlesen eines double-Wertes:

```
String filename="bsp1.grf";
FileInputStream fin;
DataInputStream din;
double d;

try {
   din = new FileInputStream(filename);
   din = new DataInputStream(fin);

   d = din.readDouble();
   din.close();
}

catch (IOException e) {
   System.err.println("IOException: " + e);
}
```

Einlesen eines integer-Wertes:

```
String filename="bsp1.grf";
FileInputStream fin;
DataInputStream din;
int i;
try {
   fin = new FileInputStream(filename);
   din = new DataInputStream(fin);

   i = din.readInt();
   din.close();
}
catch (IOException ee) {
   System.err.println("IOException: " + ee);
}
```

Klasse ArrayList:

Die Klasse ArrayList speichert beliebige Objekte in einer Liste (Vergleichbar mit einem dynamischen Array).

Methoden des Objektes ArrayList:

add(object)	Trägt ein Objekt in die Liste ein
get (i)	Gibt das i-te Element aus der Liste aus. Der Index zählt von 0 bis n-1 !!
size()	Gibt die Anzahl der Elemente aus
clear()	Löscht alle Elemente in der Liste

Deklaration und Erzeugen eines ArrayList:

- ArrayList liste1 = new ArrayList (); // speichert beliebige Objekte
- ArrayList <Circle> liste2 = new ArrayList <Circle>(); // speichert nur Kreise

Element aus der Liste holen:

- Object obj = liste1.elementAt(2);
- Circle c = (Circle) obj; // Cast
-
- Circle c = liste2.get(2); // automatischer Cast

möglicher Beispielquellcode:

```
MyLine     linie;          // speichert ein Linienobjekt
MyLine     linie2;         // Zwischenspeicher für ein Linienobjekt
MyCircle   kreis;          // speichert ein Kreisobjekt
MyCircle   kreis2;         // Zwischenspeicher für ein Kreisobjekt
Object     obj;            // Zwischenspeicher für die Liste
ArrayList liste;             // Deklaration

linie=new MyLine();
kreis = new MyKreis();
liste = new ArrayList();       // ein dynamisches Feld wird erzeugt
liste.add(linie);          // Objekt vom Typ MyLine wird in die Liste eingetragen
liste.add(kreis);          // Objekt vom Typ MyCircle wird in die Liste eingetragen

for (int i=0; i<liste.size(); i++) {
   Object obj = liste.get(i);
   // weitere Verarbeitung
}
```